ALGORITMAANALIZI VE BÜYÜK «O» NOTASYONU

Dr. Öğr. Üyesi Nilgün Şengöz

ALGORITMALAR

- Algoritmaların Özellikleri

• Input; Girdi, bir kümedir,
• Output; Çıktı, bir kümedir (çözümdür)
• Definiteness; Kesinlik (algoritmanın adımlarının belirli olması)
• Correctness ; Doğruluk (bütün girdiler için, algoritmanın tanımlı olması
• Finiteness; Sonluluk (çalışma adımlarının sonlu olması)
• Effectiveness; Verimlilik (Her adımın ve sonuca ulaşan yolun)
• Generality; Genellenebilirlik (bir problem kümesi için)

ALGORITMALAR

- Kaba Kod (pseudocode, müsvedde kod)

Örnek: Bir kümede bulunan en büyük değeri bulan kod

```
procedure max(a1, a2, …, an: integers)
max := a1
for i := 2 to n
    if max < ai then max := ai
{max en büyük elemandır}
```

ALGORITMALAR

- Doğrusal arama (linear search): Verilen bir girdi içerisinde baştan sona kadar elemanlara teker teker bakarak bir elemanın aranması

procedure: linear_search(x: integer; a1, a2,…,an: integers)
i := 1
while (i ≤ n and x ≠ ai)
i := i + 1
if i ≤ n **then** konum := i
else konum := 0
{konum, aranan elemanın sırasını göstermektedir, şayet 0 ise aranan eleman bulunamamıştır}

ALGORITMALAR

- Şayet aranan liste sıralı ise ikili arama (binary search) daha iyi sonuç verir

procedure: binary_search(x: integer; a1, a2,…,an: integers)
i := 1 {i arama aralığının sol sınırı}
j := n {j arama aralığının sağ sınırı}

while (i < j)
begin
    m := [(i+j)/2]
    if x > a_m **then** i:= m+1
    else j := m
end
if x = a_i **then** konum := i
else konum := 0
{konum, aranan x değerinin bulunduğu konumu gösterir, şayet 0 ise aranan x değeri bulunamamıştır}

KARMAŞIKLIK (COMPLEXITY)

- Yazdığımız bir algoritmanın doğru çalıştığından emin olmakla birlikte bu algoritmayı, daha önce yazılmış ve aynı sonucu veren başka algoritmalarla karşılaştırmak isteyebilirsiniz.
- Burada teknik olarak değerlendirilecek başlıca iki başlık söz konusudur.
- Birincisi algoritmaların **bellek kullanım miktarı**, ikincisi ise algoritmaların hesaplama yapmak için harcadığı **süre**dir.
- Mesela yazdığınız bir algoritma aynı işi yapan diğer bir algoritmadan daha hızlı çalışmasına rağmen çoğu bilgisayar için bellek aşımı gerçekleştiriyorsa bu pek uygun olmayacaktır.

KARMAŞIKLIK (COMPLEXITY)

- Elbette diğer algoritmalarla karşılaştırma yapmak yerine, yazdığınız bir algoritmanın tek başına analizini yapmak da isteyebilirsiniz. Bunun için yazdığınız algoritmaları ve varsa karşılaştıracağınız algoritmaları tek tek çalıştırıp hız ve bellek testi yapabilirsiniz.
- Ama bu tahmin edebileceğiniz gibi hem zaman açısından sıkıntı yaratır hem de elde edeceğiniz veriler donanımsal ve sistemsel değişikliklerden dolayı bilimsel olmaz.(Bu gibi işlemleri performans testi olarak da düşünebiliriz)
- Bu durumda matematiksel olarak ifade edebileceğimiz, donanımsal ve sistemsel bağımlılığı olmayan bir yönteme ihtiyacımız olacaktır.
- Bu yöntemle algoritmamıza girdi olarak verilen verilerin miktarına bağlı olarak sonuçlar üretiriz. İşte elde edilen bu sonuçlar ilgili algoritmanın ***karmaşıklığı*** olarak tanımlanır. Bir algoritmanın karmaşıklığı performansını etkiler ama karmaşıklık ile performans farklı kavramlardır görüldüğü gibi.

KARMAŞIKLIK (COMPLEXITY)

- Şayet girdi küçükse hafıza (space) ve zaman (time) karmaşıklıkları çok önemli değildir.

- Örneğin n = 10, için ikili veya doğrusal arama kullanılması çok önemli değildir ama n= 2^{30} gibi eleman sayılarında aralarında yıllarca fark olabilir.

KARMAŞIKLIK (COMPLEXITY)

- Örneğin, aynı problemi çözen A ve B gibi iki farklı algoritma olsun.
- A için zaman karmaşıklığı 5,000n, ve B için $\lceil 1.1^n \rceil$ olsun.
- n = 10 için A algoritması 50,000 adımda biterken, B sadece 3 adımda bitmektedir. B çok daha iyidir denebilir mi?
- n = 1000 için A 5,000,000 adımda biterken B ise $2.5 \cdot 10^{41}$ adımda bitmektedir.
- Peki hangisi iyidir?

KARMAŞIKLIK (COMPLEXITY)

•A, büyük veri kümeleri için de kullanışlıyken B'nin kullanılamayacağı anlaşılmaktadır.

•Karmaşıklığın "büyümesi" (growth) çok daha önemlidir.

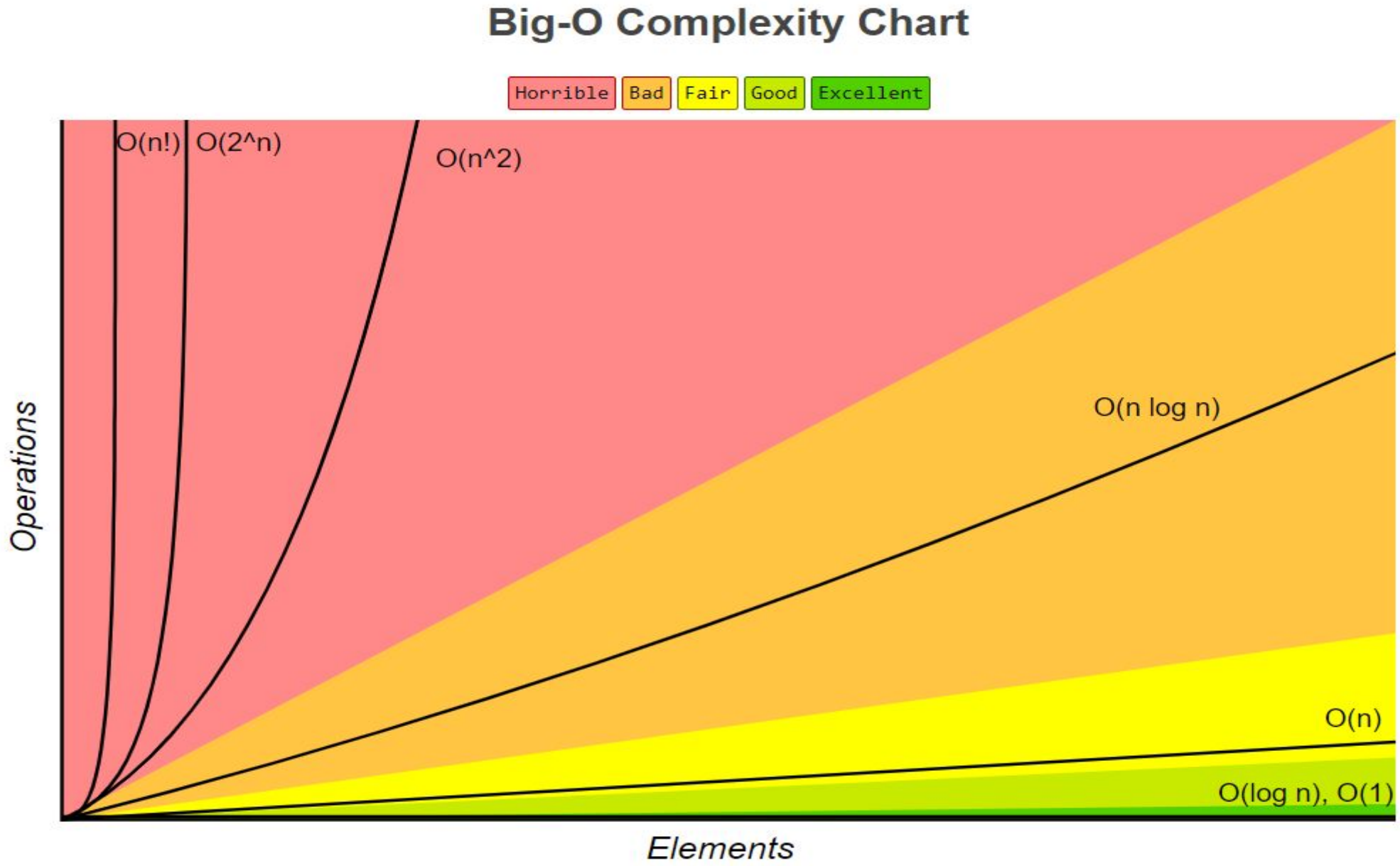
•Algoritmaların karşılaştırılması için algoritmaların zaman ve hafıza karmaşıklıklarındaki büyüme karşılaştırılır.

KARMAŞIKLIK (COMPLEXITY)

- A ve B algoritmalarının karmaşıklıkları

Girdi Boyu	Algoritma A	Algoritma B
n	5,000n	$\lceil 1.1^n \rceil$
10	50,000	3
100	500,000	13,781
1,000	5,000,000	$2.5 \cdot 10^{41}$
1,000,000	$5 \cdot 10^9$	$4.8 \cdot 10^{41392}$

KARMAŞIKLIK (COMPLEXITY)



- Büyük «O» karmaşıklık çizelgesi

KARMAŞIKLIK (COMPLEXITY)

KARŞILAŞILAN GENEL FONKSİYONLAR

Maliyet artar

İsim	Büyük-Oh	Yorum
Sabit	O(1)	Yenilmez!
Log log	O(loglogN)	Tahminsel arama
Logaritmik	O(logN)	İyi hazırlanmış arama algoritmalarının tipik zamanı
Doğrusal	O(N)	Hızlı bir algoritmadır. N tane veriyi girmek için gereken zaman.
N logN	O(NlogN)	Çoğu sıralama algoritması
Karesel	$O(N^2)$	Veri miktarı az olduğu zamanlarda uygun (N<1000)
Kübik	$O(N^3)$	Veri miktarı az olduğu zamanlarda uygun (N<1000)
Üssel	$O(2^N)$	Veri miktarı çok az olduğunda uygun (n<=20)

KARMAŞIKLIK (COMPLEXITY)

- Dizi Sıralama Algoritmaları

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)